SHALE: An Efficient Algorithm for Allocation of Guaranteed Display Advertising

Vijay Bharadwaj
Netflix[*]
bharadway.vijay@gmail.com

Peiji Chen
Yahoo!Labs
peiji@yahoo-inc.com

Wenjing Ma
Yahoo!Labs
wenjingm@yahoo-inc.com

Chandrashekhar Nagarajan
Yahoo!Labs
cn54@yahoo-inc.com

John Tomlin
opTomax Solutions[†]
johntomlin@acm.org

Sergei Vassilvitskii
Yahoo!Labs
sergei@yahoo-inc.com

Erik Vee
Yahoo!Labs
erikvee@yahoo-inc.com

Jian Yang
Yahoo!Labs
jianyang@yahoo-inc.com

ABSTRACT

Motivated by the problem of optimizing allocation in guaranteed display advertising, we develop an efficient, lightweight method of generating a compact *allocation plan* that can be used to guide ad server decisions. The plan itself uses just $O(1)$ state per guaranteed contract, is robust to noise, and allows us to serve (provably) nearly optimally. The optimization method we develop is scalable, with a small in-memory footprint, and working in linear time per iteration. It is also "stop-anytime," meaning that time-critical applications can stop early and still get a good serving solution. Thus, it is particularly useful for optimizing the large problems arising in the context of display advertising. We demonstrate the effectiveness of our algorithm using actual Yahoo! data.

1. INTRODUCTION

A key problem in display advertising is how to efficiently serve in some (nearly) optimal way. As internet publishers and advertisers become increasingly sophisticated, it is not enough to simply make serving choices "correctly" or "acceptably". Improving objective goals by just a few percent can often improve revenue by tens of millions of dollars for publishers, as well as improving advertiser or user experience. Serving needs to be done in such a way that we maximize the potential for users, advertisers, and publishers.

In this paper, we address serving display advertising in the guaranteed display marketplace, providing a lightweight optimization framework that allows real servers to allocate ads efficiently and with little overhead. Recall that in guaranteed display advertising, advertisers may target particular types of users visiting particular types of sites over a specified time period. Publishers guarantee to serve their ad some promised number of times to users matching the advertiser's criteria over the specified duration. We refer to this as a *contract*.

In [7], the authors show that given a forecast of future inventory, it is possible to create an optimal *allocation plan*, which consists of labeling each contract with just $O(1)$ additional information. Since it is so compact, this allocation plan can efficiently be communicated to ad servers. It requires no online state, which removes the need for maintaining immediately accessible impression counts. (An *impression* is generated whenever there is an opportunity to display an ad somewhere on a web page for a user.) Given the plan, each ad server can easily decide which ad to serve each impression, even when the impression is one that the forecast never predicted. The delivery produced by following the plan is nearly optimal. Note that simply using an optimizer to find an optimal allocation of contracts to impressions would not produce such a result, since the solution is too large and does not generalize to unpredicted outputs.

The method to generate the allocation plan outlined in [7] relies on the ability to solve large, non-linear optimization problems; it takes as input a bipartite graph representing the set of contracts and a sample of predicted user visits, which can have hundreds of millions of arcs or more. There are commercially available solvers that can be used to create allocation plans. However, they have several drawbacks. The most prominent of these is that such solvers aim towards finding good primal solutions, while the allocation plan generated is not directly tied to the quality of such solutions. (The allocation plan relies on the dual solution of the problem.) In particular, there is no guarantee of how close to optimal the allocation plan really is. Hence, although creating a good allocation plan is time critical, stopping the optimizer early with sub-optimal values can have undesirable effects for serving.

For our particular problem, the graph we wish to optimize is extremely large and scalability becomes a real concern. For this reason, and given the other disadvantages of using complex third party software, we propose a new solution, called 'SHALE.' It addresses all of these concerns, having many desirable properties:

- It has the "stop anytime" property. That is, after completing any iteration, we can stop SHALE and produce a good answer.
- It is a multi-pass streaming algorithm. Each iteration of SHALE runs as a streaming algorithm, reading the arcs off disk one at a time. The total online memory is proportional to the number of contracts and samples used, and is independent of the number of edges in the graph. Because of this, it is possible to handle inputs that are prohibitively large for many commercial solvers without special modifications.
- It is guaranteed to converge to the true optimal solution if it

[*] Work done while at Yahoo!Labs
[†] Work done while at Yahoo!Labs

runs for enough iterations. It is robust to sampling, so the input can be generated by sampling rather than using a full input.

- Each contract is annotated with just $O(1)$ information, which can be used to produce nearly optimal serving. Thus, the solution generated creates a practical allocation plan, useable in real serving systems.

The SHALE solver uses the idea of [7] as a starting point, but it provides an additional twist that allows the solver to stop after any number of iterations and still produce a good allocation plan. For this reason, SHALE is often five times faster than solving the full problem using a commercial solver.

1.1 Related Work

The allocation problem facing a display advertising publisher has been the subject of increased attention in the past few years. Often modeled as a special version of a stochastic optimization, several theoretical solutions have been developed [4, 6]. A similar formulation of the problem was done by Devanur and Hayes [2],who added an assumption that user arrivals are drawn independently and identically from some distribution, and then proceed to develop allocation plans based on the learned distribution. In contrast, Vee et al. [7] did not assume independence of arrivals, but require the knowledge of the user distributions to formulate the optimization problem.

Bridging the gap between theory and practice, Feldman et al. [3] demonstrated that primal-dual methods can be effective for solving the allocation problem. However, it is not clear how to scale their algorithm to instances on billions of nodes and tens of billions of edges. A different approach was given by Chen et al. [1] who used the *structure* of the allocation problem to develop control theory based methods to guide the online allocation and mitigate the impact of potential forecast errors.

Finally, a crucial piece of all of the above allocation problems is the underlying optimization function. Ghosh et al [5] define representative allocations, which minimize the average ℓ_2^2 distance between an allocation given to a specific advertiser, and the ideal one which allocates every eligible impression with equal probability. Feldman et al. [3] define a similar notion of *fair* allocations, which attempt to minimize an ℓ_1 distance between the achieved allocation and a similarly defined ideal.

2. PROBLEM STATEMENT

In this section, we begin by defining the notion of an optimal allocation of ads to users/impressions (Section 2.1). Our goal will then be to serve as close as possible to this optimal allocation. In Section 2.2, we describe the notion of generating an allocation plan, which will be used to produce nearly optimal serving.

2.1 Optimal Allocation

In guaranteed display advertising, we have a large number of forecast impressions together with a number of contracts. These contracts specify a *demand* as well as a target; we must deliver a number of impressions at least as large as the specified demand, and further, each impression must match the target specified by the contract. We model this as a bipartite graph. On one side are *supply nodes*, representing impressions. On the other side are *demand nodes*, representing contracts. We add an arc from a given supply node to a given demand node if and only if the impression that the supply node represents is *eligible* (i.e. matches the target profile) for the contract represented by the demand node. Further, demand nodes are labeled with a *demand*, which is precisely the amount of impressions guaranteed to the represented contract. In general, supply nodes will represent several impressions each, thus each supply node is labeled with a weight s_i, leading to a weighted graph (see [7] for more details). Figure 1 shows a simple example.

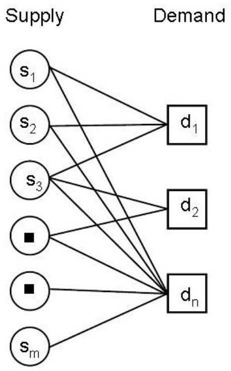

Figure 1: Example bipartite graph

An optimal allocation must both be feasible and minimize some objective function. Here, our objective balances two goals: minimizing penalty, and maximizing *representativeness*. Each demand node/contract j has an associated penalty, p_j. Let u_j be the *underdelivery*, i.e. the number of impressions delivered less than d_j. Then our total penalty is $\sum_j p_j u_j$.

Representativeness is a measure of how close our allocation is to some target. For each impression i and contract j, we define a target, θ_{ij}. In this paper, we set $\theta_{ij} = d_j/S_j$, where $S_j = \sum_{i \in \Gamma(j)} s_i$, the total eligible supply for contract j. This has the effect of aiming for an equal mix of all possible matching impressions. (Here, $\Gamma(j)$ is the neighborhood of j, likewise, we denote the neighborhood of i by $\Gamma(i)$.) The non-representativeness for contract j is the weighted L_2 distance from the target θ_{ij} and the proposed allocation, x_{ij}. Specifically,

$$\frac{1}{2} \sum_{i \in \Gamma(j)} s_i \frac{V_j}{\theta_{ij}} (x_{ij} - \theta_{ij})^2,$$

where V_j is the relative priority of the contract j; a larger V_j means that representativeness is more important. Notice that we weight by s_i to account for the fact that some sample impressions have more weight than others. Representativeness is key for advertiser satisfaction. Simply giving an advertiser the least desirable type of users (say, three-year-olds with a history of not spending money) or attempting to serve out an entire contract in a few hours decreases long-term revenue by driving advertisers away. See [5] for more discussion on this idea.

Given these goals, we may write our optimal allocation in terms of a convex optimization problem:

$$\begin{array}{llll}
\text{Minimize} & \frac{1}{2}\sum_{j, i \in \Gamma(j)} s_i \frac{V_j}{\theta_{ij}} (x_{ij} - \theta_{ij})^2 + & \sum_j p_j u_j & \\
\text{s.t.} & \sum_{i \in \Gamma(j)} s_i x_{ij} + u_j \geq d_j & \forall j & (1) \\
 & \sum_{j \in \Gamma(i)} x_{ij} \leq 1 & \forall i & (2) \\
 & x_{ij}, u_j \geq 0 & \forall i, j & (3)
\end{array}$$

Constraints 1 are called *demand constraints*. They guarantee that u_j precisely represents the total underdelivery to contract j. Constraints 2 are *supply constraints*, and they specify that we serve no more than one ad for each impression. Constraints 3 are simply *non-negativity constraints*.

The *optimal allocation* for the guaranteed display ad problem is the solution to the above problem, where the input bipartite graph represents the full set of contracts and the *full set of impressions!* Of course, generating the full set of impressions is impossible in practice. The work of [7] shows that using a sample of impressions still produces an approximately optimal fractional allocation. We interpret the fractions as the *probabilities* that a given impression should be allocated to a given contract. Since there are billions of impressions, this leads to serving that is nearly identical.

Although this paper focuses on the above problem, we note that our techniques can be extended to more general objectives. For example, in related work, [8] described a multi-objective model for the allocation of inventory to guaranteed delivery, which combined penalties and representativeness (as above) with revenue made on the non-guaranteed display (NGD) spot market and the potential revenue gained from supplying clicks to contracts. SHALE can easily be extended to handle these variants.

2.2 Compact Serving

In the previous subsection, we defined the notion of optimal allocation. However, serving such an allocation is itself a different problem. Following [7], we define the problem of online serving with forecasts as follows.

We are given as input a bipartite graph, as described in the previous subsection. (We assume this graph is an approximation of the future inventory, although it is not necessary for this definition.) We proceed in two phases.

- **Offline Phase**: Given the bipartite graph as input, we must annotate each demand node (corresponding to a contract) with $O(1)$ information. This information will guide the allocation during the online phase.
- **Online Phase**: During the online phase, impressions arrive one at a time. For each impression, we are given the set of eligible contracts, together with the annotation computed during the offline phase of each returned contract. Using only this information, we must decide which contract to serve to the impression.

The *online allocation* is the actual allocation of impressions to contracts given during the online phase. Our goal is to produce an online allocation that is as close to optimal as possible.

Remarkably, the work of [1] shows that there is an algorithm that solves the above problem nearly optimally. If the input bipartite graph exactly models the future impressions, then the online allocation produced is optimal. If the input bipartite graph is generated by sampling from the future, then the online allocation produced is provably approximately optimal.

However, the previous work simply assumed that an optimal solution can be found during the Offline Phase. Although this is true, it does not address many of the practical concerns that come with solving large-scale non-linear optimization problems. In the following sections, we describe our solution, which in addition to solving the problem of compact serving, is fast, simple, and robust.

3. ALGORITHMS

3.1 Plan creation using full solution

The proposal of [7] to create an allocation plan was to solve the problem of Section 2.1 using standard methods. From this, we can compute the *duals* of the problem. In particular, we may write the problem in terms of its Lagrangian (more formally, we use the KKT conditions). Every constraint then has a corresponding dual variable. (Intuitively, the harder a constraint is to satisfy, the larger its dual variable in the optimal solution.)

The allocation plan then consists of the demand duals of the problem, denoted α. So each contract j was labeled with the demand dual from the corresponding demand constraint, α_j. The supply duals, denoted β, and the non-negativity duals were simply thrown out.

A key insight of this earlier work is that we can reconstruct the optimal solution using only the α values. When impression i arrives, the value of β_i can be found online by solving the equation $\sum_{j\in\Gamma(i)} g_{ij}(\alpha_j - \beta_i) = 1$, resetting $\beta_i = 0$ if the solution is less than 0. Here, $g_{ij}(z) = \max\{0, \theta_{ij}(1 + z/V_j)\}$. We then set $x_{ij} = g_{ij}(\alpha_j - \beta_i)$ for each $j \in \Gamma(i)$. Somewhat surprisingly, this yields an optimal allocation. (And when the value of α is obtained by solving a sampled problem, it is approximately optimal.)

As mentioned in the introduction, although this solution has many nice properties, solving the optimization problem using standard methods is slower than desirable. Thus, we have a need for faster methods.

3.2 Greedy solution (HWM)

An alternate approach to solving the allocation problem is the *High Water Mark* (HWM) algorithm, based on a greedy heuristic. This method first orders all the contracts by their *allocation order*. Here, the allocation order puts contracts with smaller S_j (i.e. total eligible supply) before contracts with larger S_j. Then, the algorithm goes through each contract one after another, trying to allocate an equal fraction from all the eligible ad opportunities. This fraction is denoted ζ for each contract, and corresponds roughly to its demand dual. Contract j is given fraction ζ_j from each eligible impression, *unless* previous contracts have taken more than a $1-\zeta_j$ fraction already. In this case, contract j gets whatever fraction is left (possibly 0).

If there is very little contention (or contract j comes early in the allocation order), then $\zeta_j = d_j/S_j$. This will give exactly the right amount of inventory to contract j. However, if a lot of inventory has already been allocated when j is processed, its ζ_j value may be larger than this to accommodate the fact that it gets less than ζ_j for some impressions. Setting $\zeta = 1$ will give a contract all inventory that has not already been allocated. We do this in the case that there is not enough remaining inventory to satisfy the demand of j.

The pseudo-code is summarized as follows.

```
1. Order all demand nodes in decreasing contention order ($d_j/S_j$).
2. For each supply node $i$, initialize the available weight $\tilde{s_i} = s_i$.
3. For each demand node $j$, in allocation order:
   (a) Find $\zeta_j$ such that
       $$\sum_{i\in B_j} \min\{\tilde{s_i}, \zeta_j s_i\} = d_j,$$
       setting $\zeta_j = \infty$ if the above has no solution.
   (b) For each matching supply nodes $i \in B_j$
       Update $\tilde{s_i} = \tilde{s_i} - \min\{\tilde{s_i}, \zeta_j s_i\}$.
```

We note that the computation in Step 3a can be done in time linear in the size of $|B_j|$. Hence, the total runtime of the HWM algorithm is linear in the number of arcs in the graph.

3.3 SHALE

Obtaining a full solution using traditional methods is too slow (and more precise than needed), while the HWM heuristic, although very fast, sacrifices optimality. SHALE is a method that spans the two approaches. If it runs for enough iterations, it produces the true optimal solution. Running it for 0 iterations (plus an additional step at the end) produces the HWM allocation. So we can easily balance precision with running time. In our experience (see Section 4), just 10 or 20 iterations of SHALE yield remarkably good results; for serving, even using 5 iterations works quite well since forecast errors and other issues generally dwarf small variations in the solution. Further, SHALE is amenable to "warm-starts," using the previous allocation plan as a starting point. In this case, it is even better.

SHALE is based on the solution using optimal duals. The key innovation, however, is the ability to take *any* dual solution and convert it into a good primal solution. We do this by extending the simple heuristic HWM to incorporate dual values. Thus, the SHALE algorithm has two pieces. The first piece finds reasonable duals. This piece is an iterative algorithm. On each iteration, the dual solution will generally improve. (And repeated iterations converge to the true optimal.) The second piece converts the reasonable set of duals we found (more precisely, the α values, as described earlier) into a good primal solution.

The optimization for SHALE relies heavily on the machinery provided by the KKT conditions. Interested readers may find a more detailed discussion in the Appendix. Here, we note the following. If α^* and β^* are optimal dual values, then

1. The optimal primal solution is given by $x^*_{ij} = g_{ij}(\alpha^*_j - \beta^*_i)$, where $g_{ij}(z) = \max\{0, \theta_{ij}(1 + z/V_j)\}$.
2. For all j, $0 \leq \alpha^*_j \leq p_j$. Further, either $\alpha^*_j = p_j$ or $\sum_{i \in \Gamma(j)} s_i x^*_{ij} = d_j$.
3. For all i, $\beta_i \geq 0$. Further, either $\beta_i = 0$ or $\sum_{j \in \Gamma(i)} x^*_{ij} = 1$.

The pseudo-code for SHALE is shown below.

```
• Initialize. Set $\alpha_j = 0$ for all $j$.
• Stage One. Repeat until we run out of time:
    1. For each impression $i$, find $\beta_i$ that satisfies
           $\sum_{j \in \Gamma(i)} g_{ij}(\alpha_j - \beta_i) = 1$
       If $\beta_i < 0$ or no solution exists, update $\beta_i = 0$.
    2. For each contract $j$, find $\alpha_j$ that satisfies
           $\sum_{i \in \Gamma(j)} s_i g_{ij}(\alpha_j - \beta_i) = d_j$
       If $\alpha_j > p_j$ or no solution exists, update $\alpha_j = p_j$.
• Stage Two.
    1. Initialize $\tilde{s}_i = 1$ for all $i$.
    2. For each impression $i$, find $\beta_i$ that satisfies
           $\sum_{j \in \Gamma(i)} g_{ij}(\alpha_j - \beta_i) = 1$
       If $\beta_i < 0$ or no solution exists, update $\beta_i = 0$.
    3. For each contract $j$, in allocation order, do:
        (a) Find $\zeta_j$ that satisfies
                $\sum_{i \in \Gamma(j)} \min\{\tilde{s}_i, s_i g_{ij}(\zeta_j - \beta_i)\} = d_j,$
            setting $\zeta_j = \infty$ if there is no solution.
        (b) For each impression $i$ eligible for $j$, update $\tilde{s}_i = \tilde{s}_i - \min\{\tilde{s}_i, s_i g_{ij}(\zeta_j - \beta_i)\}$.
• Output The $\alpha_j$ and $\zeta_j$ values for each $j$.
```

Our implementation of SHALE runs in linear time (in the number of arcs in the input graph) per iteration.

During Stage One, we iteratively improve the α values by assuming that the β values are correct and solving the equation for α. Recall that $x_{ij} = g_{ij}(\alpha_j - \beta_i)$. Thus, we are simply solving the equation $\sum_{i \in \Gamma(j)} s_i x_{ij} = d_j$ for α_j. In order to find better β values, we assume the α is correct and solve for β using $\sum_{j \in \Gamma(i)} x_{ij} = 1$. The following theorem shows that this simple iterative technique converges, and yields an ε approximation in polynomial steps.

More precisely, define $d_j(\alpha) = \sum_{i \in \Gamma(j)} s_i g_{ij}(\alpha_j - \beta_i)$, where β is determined as in Step 1 of Stage One of SHALE. (We think of this as the projected delivery for contract j using only Stage One of SHALE.) We say a given α solution produces an *ε-approximate delivery* if for all j, either $\alpha_j = p_j$ or $d_j(\alpha) \geq (1 - \varepsilon) d_j$. Note that an optimal α_j is at most p_j; the intuitive reason for this is that growing α_j any larger will cause the non-representativeness of the contract's delivery to be even more costly than the under-delivery penalty. Thus, an ε-approximate delivery means that every contract is projected to deliver within ε of the desired amount, or its α_j is "maxed-out."

We can now state our theorem. Its proof is in the appendix.

THEOREM 1. *Stage One of SHALE converges to the optimal solution of the guaranteed display allocation problem. Further, let $\varepsilon > 0$. Then within $\frac{1}{\varepsilon} n \max_j\{p_j/V_j\}$ iterations, the output α produces an ε-approximate delivery.*

Note that Stage One is effectively a form of coordinate descent. In general, it could be replaced with any standard optimization technique that allows us to recover a set of approximate dual values. However, the form we use is simple to understand, use, and debug. Further, it works very well in practice.

In Stage Two, we calculate ζ values in a way similar to HWM. We calculate β values based on the α values generated from Stage One. Using these, we calculate ζ values to give d_j allocation (if possible) to each contract. Notice that in Stage Two, we must be cognizant of the actual allocation. Thus, we maintain a remaining fraction left, $\tilde{s}_i$, that we cannot exceed. Thus, contracts allocated latest may not be able to get the full amount specified by g_{ij}, if the fraction taken from impression i is too great.

We note that in our actual implementation, we use a two-pass version of Stage Two. In the first pass, we bound ζ_j by α_j for each j. In the second pass, we find a second set of ζ values (with no upper bounds), utilizing any left-over inventory. This is somewhat "truer" to the allocation produced by SHALE in Stage One, and gives slightly better online allocation.

3.3.1 Online Serving with SHALE

Recall that SHALE produces two values for each contract j, namely α_j and ζ_j. Given impression i, the α values for eligible contracts are used to calculate the β_i value, which is used together with the ζ values to produce the allocation. The pseudo-code is below.

Input: Impression i and the set of eligible contracts.

1. Set $\tilde{s}_i = 1$ and find β_i such that

$$\sum_{j \in \Gamma(i)} g_{ij}(\alpha_j - \beta_i) = 1$$

If $\beta_i < 0$ or no solution exists, set $\beta_i = 0$.
2. For each matching contract j, in allocation order, compute $x_{ij} = \min\{\tilde{s}_i, g_{ij}(\zeta_i - \beta_i)\}$ and update $\tilde{s}_i \leftarrow \tilde{s}_i - x_{ij}$.
3. Select contract j with probability x_{ij}. (If $\sum_{j\Gamma(i)} x_{ij} < 1$, then there is some chance that no contract is selected.)

4. EXPERIMENTS

We have implemented both the HWM and SHALE algorithms described in Section 3 and benchmarked their performance against the full solution approach (known hereafter as XPRESS) on historical booked contract sets. We have extensively tuned the parameters for XPRESS, so it is much faster than just using it "off-the-shelf." First we describe these datasets and our chosen performance metrics and then present our evaluation results.

4.1 Experimental setup

In order to test the "real-world" performance of all three algorithms we considered 6 sets of real GD contracts booked and active in the recent past. In particular, we chose three periods of time, each for one to two weeks, and two ad positions LREC and SKY for each of these time periods.

We considered US region contracts booked to the aforementioned positions and time periods and also excluded all frequency capped contracts and all contracts with time-of-day and other custom targets. Also, all remaining contracts that were active for longer than the specified date ranges were truncated and their demands were proportionally reduced. Next, we generated a bipartite graph for each contract set as in Figure 1; by sampling 50 eligible impressions for each contract in the set. This sampling procedure is described in detail in [7]. We then ran HWM, SHALE and XPRESS on each of the 6 graphs and evaluated the following metrics.

1. **Under-delivery Rate** : This represents the total under-delivered impressions as a proportion of the booked demand, i.e.,

$$U = \frac{\sum_j u_j}{\sum_j d_j} \tag{4}$$

2. **Penalty Cost** : This represents the penalty incurred by the publisher for failing to deliver the guaranteed number of impressions to booked GD contracts. Note that the true long-term penalty due to under-delivery is not known since we cannot easily forecast how an advertiser's future business with the publisher will change due to under-delivery on a booked contract. Here we define the total penalty cost to be

$$P = \sum_j p_j u_j \tag{5}$$

where u_j is the number of under-delivered impressions to contract j and p_j is the cost for each under-delivered impression. For our experiments, we set p_j to be $p_j = 0.005 + q_j$ where q_j is the revenue per delivered impression from contract j. Indeed, it is intuitive and reasonable to expect that contracts that are more valuable to the advertiser incur larger penalties for under-delivery. The offset (here $\$5CPM$) serves to ensure that our algorithms attempt to fully deliver even the contracts with low booking prices.

3. **L2 Distance** : This metric shows how much the generated allocation deviates from a desired allocation (for example a perfectly representative one). In particular, the L2 distance is the non-representativeness function $\frac{1}{2}\sum_{i\in\Gamma(j)} s_i \frac{V_j}{\theta_{ij}}(x_{ij} - \theta_{ij})^2$, the first term of the objective function in Section 2, corresponding to the weighted ℓ_2^2 distance between target and allocation.

4.2 Experiment 1

As we mentioned earlier, SHALE was designed to provide a trade-off between the speed of execution of HWM and the quality of solutions output by XPRESS. Accordingly in our first experiment we measured the performance of SHALE (run for 0, 5, 10, 20 and 50 iterations) as compared to XPRESS against our chosen metrics. Since SHALE at 0 iterations is the same as HWM, we label it as such. Figure 2 shows the penalty cost, under-delivery rate, L2 distance and completion for HWM and SHALE run for 5, 10, 20 and 50 iterations respectively as a percentage of the corresponding metric for XPRESS, averaged over our 6 chosen contract sets. Note that the y-axis labels for the under-delivery rate and penalty cost are on the left, while the labels for the L2 distance and completion time are on the right.

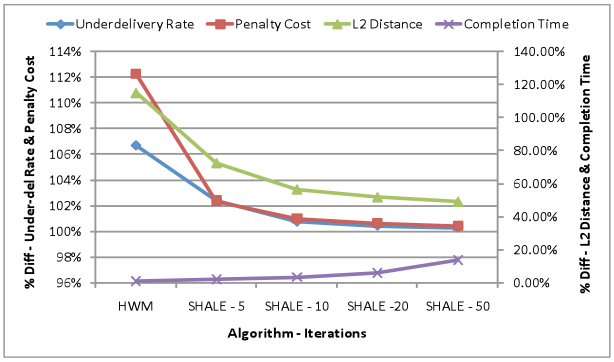

Figure 2: Performance Vs. Completion time

It is immediately clear that SHALE after only 10 iterations is within 2% of XPRESS with respect to penalty cost and under-delivery rate. Further, note that SHALE after 10 iterations is able to provide an allocation whose L2 distance is less than half that of XPRESS. (Recall smaller L2 distance means the solution is more representative, so SHALE is doing twice as well on this metric.) This somewhat surprising result seems to be an artifact of the SHALE algorithm: The functional form of g_{ij} is determined by the representativeness objective, so we can think of representativeness as "driving" the algorithm.

Even at 50 iterations, SHALE is more than 5 times as fast as XPRESS. Remarkably, its penalty and under-delivery are almost equal to XPRESS (less than 1% different), yet the L2 distance is still much better. At 20 iterations, we see SHALE gives a very high-quality solution, despite being about an order of magnitude faster than the commercial solver.

4.3 Experiment 2

We next study how SHALE performs compared to the optimal algorithm when used to serve real world sampled impressions from actual server logs. This experiment uses real contracts and real adserver logs (downsampled) for performing the complete offline simulation.

4.3.1 Setup

Here we take three new datasets which consists of real guaranteed delivery contracts from Yahoo! active during different one to two week periods in the past year. We run our optimization algorithms and serve real downsampled serving logs for each of the one-to-two week periods, reoptimizing every two hours. That is, the offline optimizer creates an allocation plan to serve the contracts for the remaining duration; we serve for two hours using that plan; collect the delivery stats so far; then re-optimize for the rest of the duration using the updated stats. Note that the two-hours corresponds to two hours of serving logs. Our actual simulation is somewhat faster due to the downsampling.

4.3.2 Algorithms compared

At the end of the simulation, we look at the contracts that start and end within the simulation period and compare how metrics of under-delivery and penalty across HWM, SHALE and DUAL algorithms. Our DUAL solution is obtained by running a coordinate gradient descent algorithm till convergencence; if our forecasts had been perfect, this would have produced optimal delivery. The SHALE algorithms are run with setting of 0, 5, 10 and 20 iterations, with the 0-iteration version labeled as HWM.

We performed serving using the reconstruction algorithm described in Section 3.3.1.

4.3.3 Metrics

The metrics include the underdelivery metric and penalty metrics as defined in Equation 4 and in Equation 5 For these set of experiments, we set p_j to be $p_j = 0.002 + 4 * q_j$ where q_j is the revenue per delivered impression from contract j.

We also compare another metric called *pacing* between these algorithms. This captures how representative contracts are with respect to time during the delivery of these contracts. The *linear goal* of a contract at a given time is the amount of delivery was perfectly smooth with respect to time. For example, a 7 day contract with demand of 14 million has a linear goal of 6 milion on day 3. In this experiment, pacing is defined as the percentage of contracts that are within 12% of the linear delivery goal at least 80% of their active duration.

4.3.4 Results

Figures 3, 4 and 5 show that the under-delivery and penalty cost for HWM (SHALE with 0 iterations) algorithm is the worst. Further, as the number of SHALE iterations increase it gets very close to the DUAL algorithm. Note that even SHALE with 5 or 10 iterations performs as well or sometimes slightly better than the DUAL algorithm. This can be attributed to different reasons; one being the fact that there are forecasting errors intrinsic to using real serving logs. Another contributing factor is the fact that the DUAL algorithm does not directly optimize for either of these metrics. In addition, Stage Two attempts to fulfill the delivery of every contract, even if it is not optimal according to the objective function. This heuristic aspect of SHALE actually appears to aid in its performance when judged by simple metrics like delivery.

Figure 6 shows how these algorithm perform with respect to pacing. The pacing is similar for all three datasets for SHALE with 5, 10 and 20 iterations when compared with the DUAL algorithm. Surprisingly, HWM has better pacing than SHALE and DUAL for two of the datasets. One possible reason for this is that SHALE and DUAL algorithm gives better under-delivery and penalty cost, compromising some pacing. Note that the time dimension is just one of the many dimensions that the representativeness portion of the objective function. This may also be an artifact of forecasting errors. In real systems, certain additional modifications are employed to ensure good pacing. For these experiments, we have removed those modifications to give a clearer picture of how the base algorithms perform.

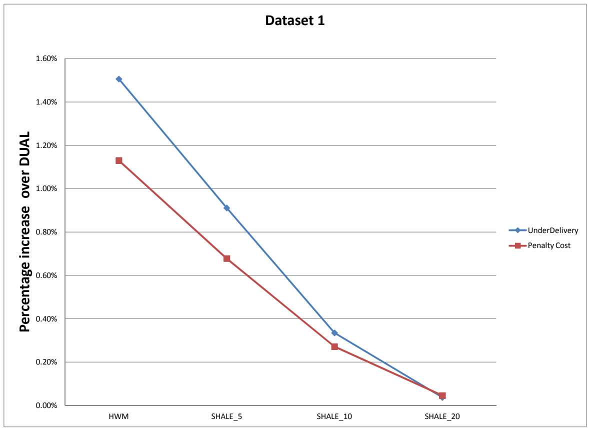

Figure 3: Dataset 1: Under Delivery and Penalty Cost Comparison

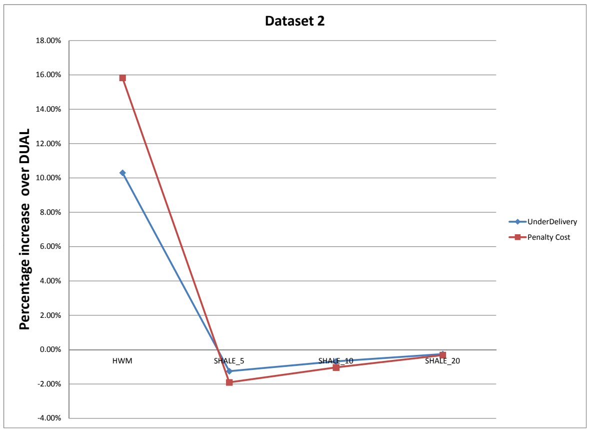

Figure 4: Dataset 2: Under Delivery and Penalty Cost Comparison

4.4 Experiment 3

Superficially, HWM and SHALE both perform well. In this experiment, we do a more detailed simulation to compare HWM and SHALE. We fix the iteration count for SHALE at 20 and test its performance under varying supply levels. Specifically, for each of our 6 contract sets, we artificially reduced the supply weight on each of the supply nodes while keeping the graph structure fixed in order to simulate the increasing scarcity of supply. We define the average supply contention (ASC) metric to represent the scarcity of supply, as follows

$$\text{ASC} = \frac{\sum_i s_i \left(\sum_{j \in i} \frac{d_j}{S_j} \right)}{\sum_i s_i} \tag{6}$$

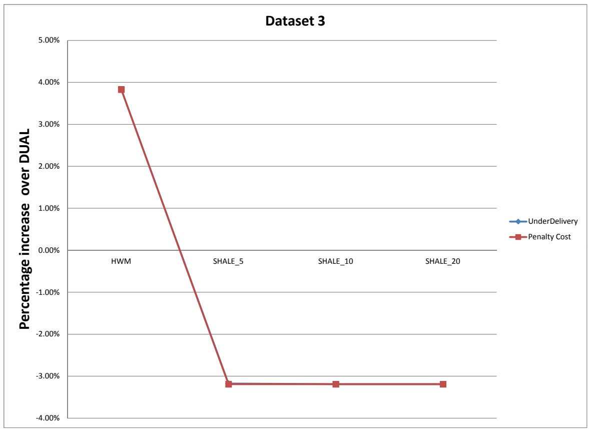

Figure 5: Dataset 3: Under Delivery and Penalty Cost Comparison

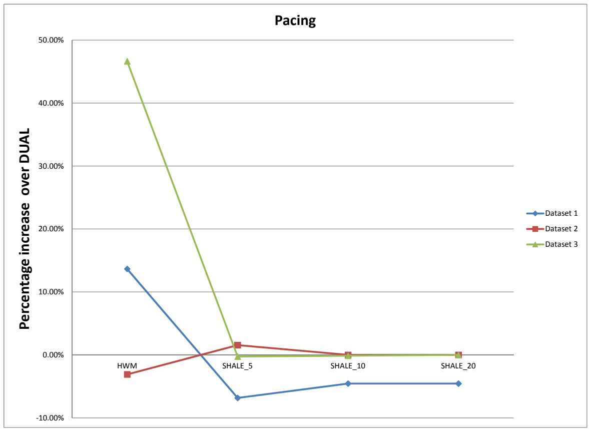

Figure 6: Pacing Comparisons on all three datasets

where s_i represents the supply weight and d_j and S_j represent the demand and eligible supply for contract j. In Figure 7, we show the under-delivery rate, penalty cost and L2 distance for SHALE as a percentage of the corresponding metric for HWM for various levels of ASC. First we note that each of our metrics for SHALE is better than the corresponding metric for HWM for all values of ASC. Indeed, the SHALE L2 distance is less than 50% of that for HWM. Also note that the SHALE penalty cost consistently improves compared to HWM as the ASC increases. This indicates that even though HWM appears to have better pacing for some data sets, SHALE is still a more robust algorithm and is likely preferrable in most situations. (Indeed, we see very consistently that its under-delivery penalty and revenue are both clearly better.)

5. CONCLUSION

We described the SHALE algorithm, which is used to generate compact allocation plans leading to near-optimal serving. Our algorithm is scalable, efficient, and has the stop-anytime property, making it particularly useful in time-sensitive applications. Our experiments demonstrate that it is many times faster than using commercially available general purpose solvers, while still leading to near-optimal solutions. On the other side, it produces a much better and more robust solution than the simple HWM heuristic. Due to its stop-anytime property, it can be configured to give the desired tradeoff between running time and optimality of the solution. Furthermore, SHALE can handle "warm starts," using a previous allocation plan as a starting point for future iterations.

SHALE is easily modified to handle additional goals, such as maximizing revenue in the non-guaranteed market or click-through rate of advertisement. In fact, the technique appears to be amenable to other classes of problems involving many users with supply constraints (e.g. each user is shown only one item). Thus, although SHALE is particularly well-suited to optimizing guaranteed display ad delivery, it is also an effective lightweight optimizer. It can handle huge, memory-intensive inputs, and the underlying techniques we use provide a useful method of mapping non-optimal dual solutions into nearly optimal primal results.

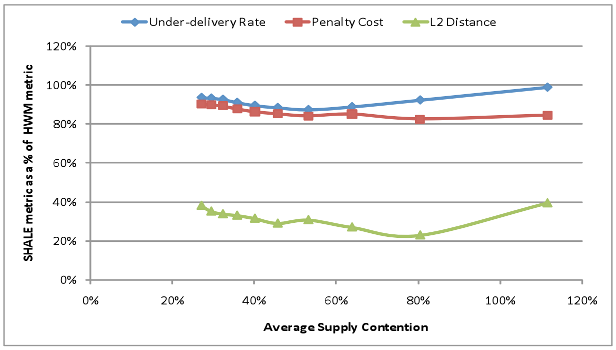

Figure 7: SHALE Vs. HWM

6. ADDITIONAL AUTHORS

7. REFERENCES

[1] Y. Chen, P. Berkhin, B. Anderson, and N. R. Devanur. Real-time bidding algorithms for performance-based display ad allocation. In C. Apté, J. Ghosh, and P. Smyth, editors, *KDD*, pages 1307–1315. ACM, 2011.

[2] N. R. Devenur and T. P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In J. Chuang, L. Fortnow, and P. Pu, editors, *ACM Conference on Electronic Commerce*, pages 71–78. ACM, 2009.

[3] J. Feldman, M. Henzinger, N. Korula, V. S. Mirrokni, and C. Stein. Online stochastic packing applied to display ad allocation. In M. de Berg and U. Meyer, editors, *ESA (1)*, volume 6346 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2010.

[4] J. Feldman, A. Mehta, V. S. Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *FOCS*, pages 117–126. IEEE Computer Society, 2009.

[5] A. Ghosh, P. McAfee, K. Papineni, and S. Vassilvitskii. Bidding for representative allocations for display advertising. In S. Leonardi, editor, *WINE*, volume 5929 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 2009.

[6] V. S. Mirrokni, S. O. Gharan, and M. Zadimoghaddam. Simultaneous approximations for adversarial and stochastic online budgeted allocation. In D. Randall, editor, *SODA*, pages 1690–1701. SIAM, 2012.

[7] E. Vee, S. Vassilvitskii, and J. Shanmugasundaram. Optimal online assignment with forecasts. In D. C. Parkes, C. Dellarocas, and M. Tennenholtz, editors, *ACM Conference on Electronic Commerce*, pages 109–118. ACM, 2010.

[8] J. Yang, E. Vee, S. Vassilvitskii, J. Tomlin, J. Shanmugasundaram, T. Anastasakos, and O. Kennedy. Inventory allocation for online graphical display advertising. *CoRR*, abs/1008.3551, 2010.

Appendix

Recall that our optimization problem is

$$\text{Minimize} \quad \tfrac{1}{2}\textstyle\sum_{j,i\in\Gamma(j)} s_i \frac{V_j}{\theta_{ij}}(x_{ij}-\theta_{ij})^2 + \displaystyle\sum_j p_j u_j$$

$$\text{s.t.} \quad \textstyle\sum_{i\in\Gamma(j)} s_i x_{ij} + u_j \geq d_j \qquad \forall j \tag{7}$$

$$s_i \textstyle\sum_{j\in\Gamma(i)} x_{ij} \leq s_i \qquad \forall i \tag{8}$$

$$x_{ij}, u_j \geq 0 \qquad \forall i,j \tag{9}$$

Notice that we have multiplied the supply constraints by s_i to aid our mathematics later.

The KKT conditions generalize the somewhat more familiar Lagrangian. Let α_j denote the demand duals. Let β_i denote the supply duals. Let γ_{ij} denote the non-negativity duals for x_{ij}, and let ψ_j denote the non-negativity dual for u_j. For our problem, the KKT conditions tell us the optimal primal-dual solution must satisfy the following

Stationarity:

$$\text{For all } i,j,\ s_i\frac{V_j}{\theta_{ij}}(x_{ij}-\theta_{ij}) - s_i\alpha_j + s_i\beta_i - \gamma_{ij}$$

$$\text{For all } i,\ p_j - \alpha_j - \psi_j = 0$$

Complementary slackness:

For all j, either $\alpha_j = 0$ or $\sum_{i\in\Gamma(j)} s_i x_{ij} + u_j = d_j$.

For all i, either $\beta_i = 0$ or $\sum_{j\in\Gamma(i)} s_i x_{ij} = s_i$.

For all i, j, either $\gamma_{ij} = 0$ or $x_{ij} = 0$.

For all j, either $\psi_j = 0$ or $u_j = 0$.

The dual feasibity conditions also tell us that $\alpha_j \geq 0$, $\beta_i \geq 0$, $\gamma_{ij} \geq 0$, and $\psi_j \geq 0$ for all i, j. (While the primal feasibility conditions tell us that the constraints in the original problem must be satified.) Since our objective is convex, and primal-dual solution satisfying the KKT conditions is in fact optimal.

Notice that the stationarity conditions are effectively like taking the derivative of the Lagrangian. The first of these tells us that

$$x_{ij} = \theta_{ij}(1 + \frac{\alpha_j - \beta_i + \gamma_{ij}/s_i}{V_j})$$

The complementary slackness condition for the γ_{ij} tells us that $\gamma_{ij} = 0$ unless $x_{ij} = 0$. This has the effect that when the expression $\theta_{ij}(1 + \frac{\alpha_j - \beta_i}{V_j})$ is negative, γ_{ij} will increase just enough to make $x_{ij} = 0$. In particular, this implies

$$x_{ij} = \max\{0, \theta_{ij}(1 + \frac{\alpha_j - \beta_i}{V_j})\} = g_{ij}(\alpha_j - \beta_i)$$

The second stationarity condition shows $\alpha_j = p_j - \psi_j$. Since $\psi_j \geq 0$, this immediately shows that $\alpha_j \leq p_j$. Further, the complementary slackness condition for ψ_j implies that $\psi = 0$ unless $u_j = 0$. That is, either $\alpha_j = p_j$ or $\sum_{i\in\Gamma(j)} s_i x_{ij} \geq d_j$. By complementary slackness of α_j, we see in fact that equality must hold (i.e. $\sum_{i\in\Gamma(j)} s_i x_{ij} = d_j$) unless $\alpha_j = 0$. But when $\alpha_j = 0$, inspection reveals that $\sum_{i\in\Gamma(j)} s_i x_{ij} = \sum_{i\in\Gamma(j)} s_i g_{ij}(-\beta_i) \leq d_j$. Hence, even when $\alpha_j = 0$, equality must hold for an optimal α_j.

Finally, the complementary slackness condition on β_i implies either $\beta_i = 0$ or $\sum_{j\in\Gamma(i)} x_{ij} = 1$. Putting all of this together, we see that

1. The optimal primal solution is given by $x^*_{ij} = g_{ij}(\alpha^*_j - \beta^*_i)$, where $g_{ij}(z) = \max\{0, \theta_{ij}(1 + z/V_j)\}$.
2. For all j, $0 \leq \alpha^*_j \leq p_j$. Further, either $\alpha^*_j = p_j$ or $\sum_{i\in\Gamma(j)} s_i x^*_{ij} = d_j$.
3. For all i, $\beta_i \geq 0$. Further, either $\beta_i = 0$ or $\sum_{j\in\Gamma(i)} x^*_{ij} = 1$.

as we claimed in Section 3.

Proof of Theorem 1. First, note that α_j is bounded above by p_j. We will show that α_j is non-decreasing on each iteration. Let α^t refer to the value of alpha computed during the t-th iteration, where $\alpha^0_j = 0$ for all j. We show by induction that $d_j(\alpha^t) \leq d_j$ for all $t \geq 0$. The base case follows by definition, since $\beta_i \geq 0$ for all i: $d_j(\alpha^0) \leq \sum_{i\in\Gamma(j)} s_i g_{ij}(0-0) = \sum_{i\in\Gamma(j)} s_i\theta_{ij} = d_j$.

So assume for some $t \geq 0$ that $d_j(\alpha^t) \leq d_j$ for all j. Let β^t be the value computed in Step 1 of Stage One of SHALE, given α^t. We see that

$$\begin{aligned} d_j(\alpha^t) &= \sum_{i\in\Gamma(j)} s_i g_{ij}(\alpha^t_j - \beta^t_i) \\ &= \sum_{i\in\Gamma(j)} s_i \max\{0, \theta_{ij}(1 + \frac{\alpha^t_j - \beta^t_i}{V_j})\} \end{aligned}$$

Further, by the way in which α^{t+1} is calculated (in Stage One, Step 2), we have that α^{t+1}_j must either be p_j or satisfy the following:

$$\begin{aligned} d_j &= \sum_{i\in\Gamma(j)} s_i g_{ij}(\alpha^{t+1}_j - \beta^t_i) \\ &= \sum_{i\in\Gamma(j)} s_i \max\{0, \theta_{ij}(1 + \frac{\alpha^{t+1}_j - \beta^t_i}{V_j})\} \end{aligned}$$

Using the fact that for any numbers $a \geq b$ that $\max\{0, a\} - \max\{0, b\} \leq a - b$ (which can be shown by an easy case analysis), we have

$$\begin{aligned} d_j - d_j(\alpha^t) &= \sum_{i\in\Gamma(j)} s_i \max\{0, \theta_{ij}(1 + \frac{\alpha^{t+1}_j - \beta^t_i}{V_j})\} \\ &\quad - \sum_{i\in\Gamma(j)} s_i \max\{0, \theta_{ij}(1 + \frac{\alpha^t_j - \beta^t_i}{V_j})\} \\ &\leq \sum_{i\in\Gamma(j)} s_i\theta_{ij}(\alpha^{t+1}_j - \alpha^t_j)/V_j \\ &= d_j(\alpha^{t+1}_j - \alpha^t_j)/V_j \end{aligned}$$

That is, either $\alpha^{t+1}_j = p_j$ or

$$\alpha^{t+1}_j = \alpha^t_j + V_j(1 - \frac{d_j(\alpha^t)}{d_j}) \tag{10}$$

Since $d_j(\alpha^t) \leq d_j$ by assumption, this shows that $\alpha_j^{t+1} \geq \alpha_j^t$ for each j. We must still prove that $d_j(\alpha^{t+1}) \leq d_j$. To this end, note that the β^{t+1} generated in Step 1 for the given α^{t+1} must greater than or equal to β^t, since $\alpha^{t+1} \geq \alpha^t$. That is, $\beta_i^{t+1} \geq \beta_i^t$ for all i. Thus,

$$\begin{aligned} d_j(\alpha^{t+1}) &= \sum_{i \in \Gamma(j)} s_i \max\{0, \theta_{ij}(1 + \frac{\alpha_j^{t+1} - \beta_i^{t+1}}{V_j})\} \\ &\leq \sum_{i \in \Gamma(j)} s_i \max\{0, \theta_{ij}(1 + \frac{\alpha_j^{t+1} - \beta_i^{t}}{V_j})\} \\ &= d_j \end{aligned}$$

as we wanted.

In general, we can use the fact that $d_j(\alpha^t) \leq d_j$ for all t, together with Equation 10, to see that the α_j values are non-decreasing at each iteration. From this (together with the fact that α_j is bounded by p_j), it immediately follows that the algorithm converges.

To see that the algorithm converges to the optimal solution, we note that the dual values generated by SHALE satisfy the KKT conditions at convergence: for all j, either $\alpha_j = p_j$ or $d_j(\alpha) = d_j$ (i.e. $p_j - \alpha_j - \psi_j = 0$ with either $\psi_j = 0$ or $u_j = 0$), with similar arguments holding for the other duals. Since the problem we study is convex, this shows that the primal solution generated must be the optimal.

As for our second claim, suppose that there is some j for which $\alpha_j^t \neq p_j$ but $d_j(\alpha_j^t) \leq (1 - \varepsilon)d_j$. Then by Equation 10, we see

$$\alpha_j^{t+1} = \alpha_j^t + V_j(1 - \frac{d_j(\alpha^t)}{d_j}) \geq \alpha_j^t + V_j\varepsilon$$

That is, α_j^{t+1} increases (over α_j^t) by at least εV_j. Since α_j^t starts at 0, is bounded by p_j, and never decreases, we see that this can happen at most $p_j/(\varepsilon V_j)$ times for each j. In this worst case, this happens for every j, giving us the bound we claim.

□